European Journal of Operational Research 222 (2012) 31–43

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Local search methods for the flowshop scheduling problem with flowtime minimization

Quan-Ke Pan [a,b], Rubén Ruiz [c,*]

[a] State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, PR China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Universitat Politècnica de València, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Camino de Vera S/N, 46021 Valencia, Spain

ARTICLE INFO

Article history:
Received 23 October 2011
Accepted 25 April 2012
Available online 3 May 2012

Keywords:
Scheduling
Flowshop
Flowtime
Local search
Metaheuristics

ABSTRACT

Flowshop scheduling is a very active research area. This problem still attracts a considerable amount of interest despite the sheer amount of available results. Total flowtime minimization of a flowshop has been actively studied and many effective algorithms have been proposed in the last few years. New best solutions have been found for common benchmarks at a rapid pace. However, these improvements many times come at the cost of sophisticated algorithms. Complex methods hinder potential applications and are difficult to extend to small problem variations. Replicability of results is also a challenge. In this paper, we examine simple and easy to implement methods that at the same time result in state-of-the-art performance. The first two proposed methods are based on the well known Iterated Local Search (ILS) and Iterated Greedy (IG) frameworks, which have been applied with great success to other flowshop problems. Additionally, we present extensions of these methods that work over populations, something that we refer to as population-based ILS (pILS) and population-based IG (pIGA), respectively. We calibrate the presented algorithms by means of the Design of Experiments (DOE) approach. Extensive comparative evaluations are carried out against the most recent techniques for the considered problem in the literature. The results of a comprehensive computational and statistical analysis show that the presented algorithms are very effective. Furthermore, we show that, despite their simplicity, the presented methods are able to improve 12 out of 120 best known solutions of Taillard's flowshop benchmark with total flowtime criterion.

1. Introduction

Finite capacity scheduling entails the determination of the processing order of a series of jobs that have to be processed on the available machines in a production shop. A first classification of scheduling problems can be derived according to the way machines are distributed in the factory. When several machines are arranged in series and jobs must visit all these machines in the same order we have what is called a flowshop. These problems have been subjected to detailed studies since the pioneering work of Johnson (1954). More specifically, a flowshop problem comprises a set N of n jobs that must be processed on a set M of m machines. These m machines are arranged in series and each job $j \in N$ is broken down into m tasks, one per machine. A job models a given production lot of a product or client order that must be manufactured. All jobs visit machines in the same order and p_{ij} denotes the known, non-negative and deterministic amount of time that job j needs at machine i. At any given time, a job is either waiting for processing or being processed by exactly one machine. Similarly, machines are either idle or occupied by a job. Baker (1974, Chapter 6, pp. 136–137) further details all restrictions that apply: All jobs are independent and available for processing at time 0. Machines never break down and are always ready. Once started at a machine, jobs are processed until completion with no preemption allowed, etc. A schedule is obtained after devising a permutation of the jobs for every machine, resulting in $(n!)^m$ possible solutions. The setting is usually simplified and only permutation schedules are examined, resulting in the permutation flowshop scheduling problem (PFSP) where job passing is not allowed, i.e., all jobs visit the machines in the same order. This reduces the number of solutions to $n!$ The objective in the PFSP is to find a permutation such that a given criterion is optimized. Most studied criteria are based on the completion times of the jobs at machines. More specifically, let $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ be a possible permutation or solution to the problem. The completion time of job j at position $\pi(j)$ at machine i is denoted by $C_{i,\pi(j)}$ and it is computed as follows:

$$C_{i,\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i,\pi(j)} \quad (1)$$

where $j = 1, \ldots, n$, $i = 1, \ldots, m$, $C_{i,\pi(0)} = 0$, and $C_{0,\pi(j)} = 0$.

* Corresponding author.
E-mail address: rruiz@eio.upv.es (R. Ruiz).

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.04.034

The completion time of a job j in the shop is then $C_{m,j}$ or C_j for short. With completion times, many different objectives are defined. The most studied criterion is the minimization of the makespan or $C_{\max}$, where $C_{\max} = \max_{j=1,\ldots,n} C_j$. This paper studies the total flowtime minimization, which has also been studied intensively. Total flowtime is defined as:

$$TFT = \sum_{j=1}^{n} C_j \tag{2}$$

When there are no release dates, total flowtime and total completion time are equivalent objectives. Total flowtime minimization reduces the work in progress or WIP and results in a stable utilization of resources. Jobs "stay" in the shop a reduced amount of time (Framinan et al., 2005). This is of particular importance to industries where reducing inventory or holding costs is of paramount importance.

The PFSP with total flowtime criterion is denoted as $n/m/P/\sum C_j$ or as $F/prmu/\sum C_j$ according to the well known existing scheduling notations (Pinedo, 2009, and many others). $F/prmu/\sum C_j$ has been proved to be NP-hard in the strong case for $m \geqslant 2$ after the results of Gonzalez and Sahni (1978). Although some exact methods have been reported in the literature (Ignall and Schrage, 1965; Bansal, 1977; Stafford, 1988 and others), they are limited to small problem instances as solving times quickly become impractical for realistically-sized cases. As a result, research has focused on the development of heuristics that produce reasonable solutions with low time and memory requirements. Some heuristics have been presented by Rajendran (1993), Rajendran and Ziegler (1997) and Li and Wu (2005), to name just a few. With the advent of powerful desktop computers, and now for more than two decades, special emphasis has been given to the study of metaheuristics, capable of producing near optimal solutions, albeit normally at the cost of longer calculations. Some examples are the genetic algorithm of Tang and Liu (2002), ant colony optimization (ACO) of Rajendran and Ziegler (2004) and the differential evolution of Pan et al. (2008), among many others.

Metaheuristics provide excellent results and constitute the state-of-the-art methods available for the PFSP with total flowtime criterion. However, many metaheuristics are fairly sophisticated and depend on several parameters and settings that might be problem and even instance dependent. Most of the time, the presented methods are so specifically tailored for the problem at hand that slight variations of the scheduling setting require extensive changes in the algorithms or even render them inapplicable. In some cases, published algorithms are so intricate that an independent coding is unlikely to obtain the same reported effectiveness or efficiency without contacting the authors to obtain detailed information and/or source codes. All this severely hinders potential practical applications. Therefore, simple, general and easily adaptable algorithms are highly desirable. However, such simplistic methods might produce lower quality solutions and a difficult compromise arises between simplicity and performance.

The Iterated Local Search (ILS) and Iterated Greedy (IG) frameworks, described by Lourenço et al. (2010) and Ruiz and Stützle (2007), respectively, constitute two simple templates for combinatorial optimization. They have resulted in state-of-the-art results for several problems, including the permutation flowshop. Following the successful application of the above two local search based frameworks, this paper presents four algorithms: an IGA, an ILS, and two population-based extensions, dubbed as population-based IGA (pIGA), and population-based ILS (pILS), respectively. The main focus is on simplicity, extensibility and ease of coding and replication of results. The presented methods employ some powerful, yet simple operators in order to improve performance. The results of the presented algorithms are compared to those of recently published metaheuristics. The computational results and statistical analyses show, as we will detail, that the presented algorithms are new state-of-the-art methods for the problem under consideration.

The rest of the paper is organized as follows. Section 2 reviews the literature of the PFSP with total flowtime minimization criterion. Section 3 presents the four local search based algorithms in detail. The proposed algorithms are calibrated in section 4. A comprehensive comparison of the presented algorithms is shown, along with statistical analyses, in Section 5. Finally, we conclude the paper in Section 6.

2. Literature review

The PFSP with total flowtime criterion was first studied by Ignall and Schrage (1965) and by Gupta (1972). This is more than a decade later than the pioneering work of Johnson (1954) for makespan minimization in the PFSP. Due to the difficulty faced by exact methods to solve medium size or large instances, efforts have been mainly dedicated to finding high quality solutions in a reasonable computational time by using heuristic or metaheuristic optimization techniques. Framinan et al. (2005) provide a comprehensive review and evaluation of heuristics for the PFSP with total flowtime criterion. Here we mention just the most cited heuristics. Rajendran (1993), Rajendran and Ziegler (1997), Liu and Reeves (2001), Li and Wu (2005) and, more recently, Laha and Sarin (2009) present high performing simple heuristics. Other more elaborated methods are those of Allahverdi and Aldowaisan (2002), Framinan et al. (2005), and Li et al. (2009). In any case, in order to attain a better solution quality for the problem under consideration, modern metaheuristics have been increasingly applied in recent years. One of the earliest known applications of genetic algorithms (GA) is due to Vempati et al. (1993). In this case, a simple GA was presented but only applied to small instances of size 25×10 (25 jobs and 10 machines) maximum. Later, Yamada and Reeves (1998) presented a genetic local search algorithm (GA_{LS}) providing good quality solutions for five sets of Taillard (1993) instances (20×5, 20×10, 20×20, 50×5 and 50×10) but needing large computational times. Gupta et al. (2000) designed a tabu search (TS) based approach that was compared against the heuristics of Rajendran (1993) obtaining better results for the tested instances. Rajendran and Ziegler (2004) proposed two ant colony optimization (ACO) algorithms, called M-MMAS and PACO, respectively, for makespan and total flowtime minimization. PACO showed better performance than M-MMAS and the best heuristic proposed by Liu and Reeves (2001). Later, Rajendran and Ziegler (2005) have introduced a new ACO algorithm based on similar concepts to those of M-MMAS and PACO with slightly better performance in some scenarios. Tasgetiren et al. (2007) extended a continuous particle swarm optimization (PSO) method to the PFSP with both makespan and total flowtime criteria. With this method, 57 out of 90 best known solutions reported by Liu and Reeves (2001) and Rajendran and Ziegler (2004) for Taillard (1993) benchmarks were improved. However, the PSO was soon surpassed by the combinatorial PSO (CPSO) of Jarboui et al. (2008) and also by the discrete differential evolution (DDE_{RLS}) and iterated greedy algorithms (IG_{RLS}) of Pan et al. (2008).

Quite recently, it seems that there has been an intensified interest in this problem as quite a number of new metaheuristics have been published. Tseng and Lin (2009) proposed a hybrid genetic local search algorithm (denoted as HGA_{T1}) by employing GA to do the global search and two methods, Insertion Search and Insertion Search with Cut-and-Repair, to do the local search. The authors demonstrated improved performance of their proposed HGA_{T1} over the PSO of Tasgetiren et al. (2007), GA_{LS} of Yamada and Reeves

(1998), and also M-MMAS and PACO of Rajendran and Ziegler (2004). Later, the same authors (Tseng and Lin, 2010) presented a similar genetic local search algorithm (denoted as HGA_{T2}) by using TS to do the local search. Zhang et al. (2009) proposed a hybrid genetic algorithm (HGA_Z for short) that employs a local search consisting of the RZ improvement procedure in Rajendran and Ziegler (1997) and the forward pairwise exchange (FPE) method in (Liu and Reeves, 2001). In this study, a new crossover operator is introduced by using an artificial chromosome generated from a weighted simple mining gene structure. The authors' experimental results proved that the proposed HGA_Z is a new state-of-the-art method for the problem considered. The same year, Dong et al. (2009) developed a simple ILS algorithm (denoted as ILS_D) that improves over M-MMAS, PACO (Rajendran and Ziegler, 2004) and the PSO of Tasgetiren et al. (2007) by a considerable margin. Jarboui et al. (2009) presented an estimation of distribution algorithm (EDA_J), where a variable neighborhood search (VNS_J) is used as an improvement procedure. Based on the experimental results, the authors claimed that their EDA_J outperformed all the existing techniques to minimize total flowtime for the PFSP. More recently, Zhang and Li (2011) have presented another estimation of distribution algorithm (EDA_Z) with a longest common subsequence operator being incorporated into the probability distribution model to mine good "genes". Different from more common EDAs, EDA_Z produces each offspring from a seed, which is selected from the population by the roulette method. The authors' experiments showed that EDA_Z produces better results than the EDA_J, DDE_{RLS}, HGA_{T1}, and ILS_D algorithms for the first nine set benchmarks of Taillard (1993). Zheng and Yamashiro (2010) have developed a quantum differential evolutionary algorithm (QDEA) based on the basic quantum-inspired evolutionary algorithm to minimize makespan, total flowtime, and maximum lateness of jobs for permutation flowshops, respectively. QDEA adopts differential evolution to perform the update of quantum gate and variable neighborhood search as a local search. The comparison of QDEA with M-MMAS, PACO, and the best heuristic of Liu and Reeves (2001) demonstrated its effectiveness. Tasgetiren et al. (2011) presented a discrete artificial bee colony algorithm (DBAC) and a hybrid differential evolution algorithm (hDDE) by hybridizing a variable neighborhood search procedure based on swap and insertion neighborhood structures. According to the experiments conducted by the authors, both algorithms provided better results than EDA_J and HGA_{T1}. Xu et al. (2011) presented an asynchronous genetic local search algorithm (AGA for short), where all pairs of individuals perform asynchronous evolutions with different local search methods. The computational results show that AGA outperforms several state-of-the-art methods including HGA_Z, EDA_J and VNS_J. Algorithms designed for parallel architectures have also been developed for total flowtime criterion. For example, Czapiński (2010) proposed a parallel simulated annealing with genetic enhancement algorithm providing better results than HGA_Z and HGA_{T1}. Additionally, Dubois-Lacoste et al. (2011) presented an Iterated Greedy Algorithm for the bi-objective flowshop.

As we can see, there is quite a number of high performing methods claiming state-of-the-art performance that have appeared in years 2009–2011. From the short review, it is also clear that some of these methods are intricate and are based on complex algorithmic templates. It is worth mentioning that there does not exist a comprehensive computational evaluation and comparison of these recent techniques. Therefore, from the existing isolated computational evaluations with different computers, programming languages, stopping criteria, and in some cases, even benchmarks, it is very difficult to ascertain which algorithm gives the best overall performance for the problem considered. In this paper, we recode twelve recently presented metaheuristics: DDE_{RLS} and IG_{RLS} of Pan et al. (2008), HGA_{T1} of Tseng and Lin (2009), HGA_{T2} of Tseng and Lin (2010), HGA_Z of Zhang et al. (2009), ILS_D of Dong et al. (2009), EDA_J and VNS_J of Jarboui et al. (2009), AGA of Xu et al. (2011), DABC and hDDE of Tasgetiren et al. (2011) and SLS of Dubois-Lacoste et al. (2011). We also present four simple local search based algorithms. A comparison among the algorithms is given based on the well known benchmark suite of Taillard (1993). In our opinion, finding such comprehensive and extensive tests among so many recent methods is not common in the scheduling literature and constitutes a main contribution of the present paper.

3. Proposed local search based algorithms

Iterated Local Search (ILS), presented by Lourenço et al. (2010) and iterated greedy (IG, Ruiz and Stützle, 2007), are two simple local search based metaheuristics that have resulted in top performance despite of their simplicity. In recent years, both ILS and IG have attracted much attention from researchers precisely due to their simplicity, effectiveness and efficiency. For example, ILS has already been successfully applied for solving the permutation flowshop problem with makespan criterion (Stützle, 1998b), the quadratic assignment problem (Stützle, 2006) and multiple depot vehicle scheduling (Laurent and Hao, 2009), among many other problems. For an updated review on ILS see Lourenço et al. (2010). IG has shown state-of-the-art performance for the PFSP with makespan criterion (Ruiz and Stützle, 2007), sequence dependent setup times PFSP with makespan and tardiness objectives (Ruiz and Stützle, 2008), unrelated parallel machines scheduling (Fanjul-Peyro and Ruiz, 2010), PFSP with blocking constraints (Ribas et al., 2011) and even multiobjective PFSP problems in Minella et al. (2011) or in Dubois-Lacoste et al. (2011). It is possible to find other recent applications of IG to other fields and more complex scheduling problems. For example, Urlings et al. (2010) have recently applied IG methods to solve complex hybrid flexible flowline scheduling problems with many additional constraints. In view of all these state-of-the-art results, we propose the application of the ILS and IG frameworks to the PFSP with total flowtime criterion. ILS and IG always deal with only one incumbent solution. Given the previous literature review, where many population-based genetic algorithms have been proposed, we also extend the ILS and IG frameworks to work with populations. Population-based ILS methods have been presented by Stützle (1998, 2006) as well as by many others. IG extensions are less studied (Ballestın et al., 2007). The research question is therefore if ILS and IG benefit from a pool or population of solutions. The details of the presented algorithms are given in the following sections.

3.1. Iterated Local Search algorithm

ILS is a simple and generally applicable stochastic local search method presented by Lourenço et al. (2010) for solving optimization problems. The essential idea of ILS is to perform a randomized walk in the space of local optima. ILS starts from a heuristically constructed solution to which a local search is applied. Generally, a local optimum is obtained. In order to escape from this local optimum, a perturbation in the solution is carried out and a new local optimum is found after applying local search again. Finally, an acceptance criterion is used in order to decide if the new local optimum should replace the first. The above process is repeated until a termination criterion is met. An outline of the ILS procedure is given in Fig. 1.

As we can see, ILS is extremely simple and general. All that is needed is a way of representing the solution (in our case a permutation of jobs), a heuristic to initialize the method, a local search procedure, a perturbation process and an acceptance criterion. Note that the most complex part is the local search, which is

procedure ILS
$\pi_0 \leftarrow GenerateInitialSolution$
$\pi \leftarrow LocalSearch(\pi_0)$ % Local search
repeat
$\pi' \leftarrow Perturbation(\pi)$ % Perturbation of the local optimum
$\pi'' \leftarrow LocalSearch(\pi')$ % Local search
$\pi \leftarrow Acceptcriterion(\pi'', \pi)$ % Decide if new solution replaces the incumbent
until termination criterion met
end

Fig. 1. Iterated Local Search (ILS) pseudo-code.

also needed for most other well known state-of-the-art methods. We now detail all these components.

3.1.1. Initialization method

It is common to initialize metaheuristics with high performing heuristics. According to Liu and Reeves (2001), Dong et al. (2009), Zhang et al. (2009), and Li et al. (2009), among others, the LR (x) heuristic developed by Liu and Reeves (2001) is a very effective method for the PFSP with total flowtime. LR (x) constructs x different sequences by appending jobs one by one using an index function and the sequence with the minimum flowtime is selected as the final solution. The index functions employed are weighted total machine idle time, artificial flowtime and a combined index. The procedure of LR (x) is briefly described as follows:

Step 1: Rank the jobs according to ascending order of the index function value and break ties according to an ascending order of the weighted total machine idle time value.
Step 2: Use each of the first x ranked jobs as the first job of the n resulting sequences. Complete the sequences by selecting jobs one by one according to the index function.
Step 3: Select the sequence with the minimum total flow time as the final solution.

LR(x) does not fix the number of sequences to be generated, as it can be adjusted to the requirements of the problem. So the heuristic is flexible in the computational effort required. Following Li et al. (2009) and Zhang et al. (2009), we use *LR*(n/m) to generate an initial solution for the proposed ILS algorithm.

3.1.2. Local Search procedure

The improvement procedure presented by Rajendran and Ziegler (1997) (denoted as RZ) is a typical local search method based on an insertion neighborhood, which is used in the composite heuristics of Li and Wu (2005) and Li et al. (2009), the ILS_D algorithm of Dong et al. (2009), and the HGA_Z of Zhang et al. (2009). The RZ procedure sequentially inserts each job in the seed sequence at all possible positions. The improvement scheme identifies the best position of the insertion for a given job and the resulting sequence is used to replace the current one if there is an improvement in the total flowtime value. Let $\pi^s = (\pi^s(1), \pi^s(2), \ldots, \pi^s(n))$ be a seed sequence, and π be the sequence returned by RZ. The procedure of RZ is outlined in Fig. 2.

The above RZ procedure is a single pass local search. If the starting solution is improved, there is the possibility of calling RZ again to improve the solution even further. Obviously, this increases the computational cost. Therefore, there is a trade-off between the algorithm's effectiveness (in terms of solution quality) and efficiency (in terms of computational time). Our tests indicate that RZ can be iteratively applied until a local optima is obtained, i.e., we stop the local search when the provided solution π does not change after calling RZ. We denote this iterated RZ procedure as iRZ in short. It is important to remark that our implemented RZ method implements Taillard (1990) accelerations, albeit only half of it, as one does not need to re-evaluate the part of the solution that has not changed. These accelerations basically speed up the procedure by about 45%.

3.1.3. Perturbation procedure and acceptance criterion

In order to escape from a local optimum and to explore new regions in the solution space, ILS applies a perturbation procedure to generate new starting points for the local search by modifying the current solution. The perturbation procedure in the presented ILS algorithm consists of a number γ of random insertion moves. Each one randomly selects a job from the permutation and inserts it into a different, randomly selected position. The number of insertions or Perturbation length γ is a key parameter, which has an important effect on the performance of ILS. A small γ value favors local

Procedure $RZ(\pi)$
$\pi^s \leftarrow \pi$
for $j \leftarrow 1$ **to** n **do**
$\pi' \leftarrow \pi$
Remove job $\pi^s(j)$ from π'.
Test $\pi^s(j)$ in all the possible positions of π' except for its original one.
Insert $\pi^s(j)$ in π' at the position resulting in the lowest total flowtime.
if $f(\pi') < f(\pi)$ **then** $\pi \leftarrow \pi'$
endfor
end

Fig. 2. The RZ local search procedure of Rajendran and Ziegler (1997).

exploration or intensification but may lead to a stagnation of the search due to a lower chance of escaping strong local optima. A larger γ value benefits global exploration but if γ is too high, the algorithm may behave like a random restart local search with a very low probability of finding better solutions. Therefore, a suitable γ value should be determined for the presented ILS algorithm. We calibrate the γ value by means of a Design of Experiments (DOE) approach later in section 4.

After a new local optimum is obtained, we have to decide if this new local optimum replaces the current incumbent solution. Three simple acceptance criteria are presented in Stützle (2006) including *random walk*, *better*, and *simulated annealing type*. *Random walk* accepts new solutions irrespective of its objective value resulting in a random walk over local optimum solutions. *Better* accepts new solutions only if they are better. This usually results in a premature convergence in the search due to insufficient diversification. *Simulated annealing type* is a compromise between the *random walk* and *better* criteria, and can be achieved by accepting worse solutions with a certain probability. Therefore, we consider this later criterion. As in Osman and Potts (1989), Stützle (1998b) and Ruiz and Stützle (2007, 2008), we adopt a constant temperature, which depends on the particular instance and it is computed as follows:

$$\text{Temperature} = \lambda \cdot \frac{\sum_{j=1}^{n}\sum_{i=1}^{m} p_{ij}}{10\,mn} \tag{3}$$

where λ is another parameter that needs to be adjusted. However, and as noted in Ruiz and Stützle (2007, 2008), this parameter has been shown to be very robust.

3.1.4. The procedure of the presented ILS algorithm

The proposed ILS algorithm for minimizing total flowtime in the PFSP is summarized in Fig. 3. Note that *rand*(·) is a function that returns a random number uniformly distributed in the range [0, 1].

Note that the proposed ILS is not the first one presented in the literature for the total flowtime minimization in the PFSP. As reviewed, Dong et al. (2009) developed a simple ILS algorithm, denoted as ILS_D. The main differences between the presented ILS method and ILS_D are the following: On the one hand, different acceptance criteria are used. ILS_D uses the "*better*" version which accepts new solutions only if they are better, whereas the presented ILS utilizes the simulated annealing type acceptance with a certain probability to accept worse solutions. On the other hand, we adopt the perturbation procedure consisting of several random insertion moves in the presented ILS, while several random adjacent pairwise interchanges are employed in ILS_D. Both the simulated annealing type acceptance and insertion moves help to escape from local optima and result in the presented ILS algorithm with better exploration than the ILS_D algorithm. Lastly, ILS_D employs a different local search scheme.

3.2. Population variant: the pILS algorithm

As shown, ILS works over an incumbent solution π and returns the best solution $\pi*$ after the optimization run. One possible weak spot is that this imposes a single search direction. Population-based metaheuristics, such as, for example, genetic algorithms, have been widely employed in flowshop scheduling. Therefore, we also propose a population ILS, referred to as pILS, that maintains a population of solutions during the search. However, we are concerned about keeping the proposed methods simple. Our presented pILS uses LR (*x*) to generate a population of *x* initial solutions. Instead of just using the best solution returned by *LR*(*n*/*m*), we keep all the constructed *x* sequences to form the initial population (so *x* is the population size). After initialization, pILS picks a solution from the population using a selection operator and applies the perturbation procedure presented in section 3.1.3. Then pILS performs the iRZ local search to the perturbed solution to generate a local optimum.

Two important issues arise when dealing with a population ILS method. First, at each iteration, a selection operator has to be applied in order to select promising solutions. Selecting just the best

procedure the presented ILS algorithm
  Set the parameters γ *and* λ
  $\pi_0 \leftarrow LR(n/m)$ % Generate an initial solution
  $\pi \leftarrow iRZ(\pi_0)$ % Local search until local optimum
  $\pi^* \leftarrow \pi$ % Best solution found so far
  repeat
    $\pi' \leftarrow Perturbation(\pi)$ % Perturbation of the local optimum
    $\pi'' \leftarrow iRZ(\pi')$ % Local search until local optimum
    if $\sum C_j(\pi'') < \sum C_j(\pi)$ **then** % Acceptance criterion
      $\pi \leftarrow \pi''$ % Accept if better than incumbent
      if $\sum C_j(\pi'') < \sum C_j(\pi^*)$ **then** % check if new best solution
        $\pi^* \leftarrow \pi''$
      endif
    elseif $rand() \le \exp\{(\sum C_j(\pi) - \sum C_j(\pi''))/Temperature\}$ **then**
      $\pi \leftarrow \pi''$ % Simulated annealing acceptance criterion
    endif
  until *termination criterion is met*
end

Fig. 3. Pseudo-code of the presented ILS algorithm.

solution basically nullifies the population advantage. Randomly selecting individuals results in a slow converging method. Second, once an ILS iteration has been finished, we have to decide if the new solution is accepted into the population or discarded. Diversification and intensification are two key issues in the optimization process of population-based methods. Diversification aims to maintain sufficient diversity within the population so that individuals are spread out widely within the search space (Yao et al., 2010). Ideally, a diverse population is more likely to evolve. However, as the population evolves after a number of generations, its diversity diminishes and the individuals in the population become very similar. This results in search stagnation and the best solution in the population ceases to improve. To overcome these issues, we present two enhancements. These come in the form of a bi-selection method and a diversity control mechanism.

For selection operators, tournament is widely used in evolutionary algorithm applications for PFSPs due to its simplicity. We consider a tournament selection with size two in the presented pILS. That is, two solutions are picked randomly from the current population, and the one with the lower total flowtime value is chosen. However, if only the value of total flowtime is used as the measure for selection, some promising individuals with larger total flowtime values will be eliminated soon. These individuals may lead to much better solutions after a number of iterations. Therefore, it is important to increase the probability that these individuals have in the selection. We use the "*age*" to represent the number of iterations an individual survives. Younger individuals undergo less iterations. The search areas around them are not well explored. We increase the chance of selection for these individuals, and consider another tournament selection using the age of individuals as a measure. That is, we randomly pick two individuals from the population, and the younger one is chosen for reproduction. In our pILS, the presented two selection schemes are applied randomly with equal probability (50%:50%) in the search.

We also consider the diversification of the population in the generational scheme, the process by which offspring replace old members from the previous generation. If the generated local optimum is better than the worst solution in the population, and if there is no other identical solution in the population, the obtained solution replaces the worst solution and becomes a new member of the population. This population management with clone avoidance is known as steady state and was first used for flowshop scheduling problems by Ruiz et al. (2006). However, note that two solutions might slightly differ in their respective permutations so this steady state generational scheme still suffers from population convergence. We also consider a diversity measure for the population. With this, the new solution in only included into the population if also the average diversity measure of the population does not decrease. An aspiration criterion is utilized. If the generated local optimum is strictly better than every individual of the population, the worst solution is replaced by the generated local optimum, regardless of the deterioration in the average diversity measure.

We use the diversity measure recently presented by Pan and Ruiz (2012). The measure is based on both the job order and on similar blocks of jobs in the sequences of the current population. It is now briefly explained as follows:

Step 1. Calculate the job order matrix $[\phi_{i,j}]_{n\times n}$ as $[\phi_{i,j}]_{n\times n}=\begin{bmatrix}\phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,n}\\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,n}\\ \cdots & \cdots & \ddots & \cdots\\ \phi_{n,1} & \phi_{n,2} & \cdots & \phi_{n,n}\end{bmatrix}$, where $\phi_{i,j}$ is the number of times that job j appears at position i after considering all individuals of the population.

Step 2: Calculate the block matrix $[\lambda_{j'j}]_{n\times n}$ as follows: $[\lambda_{j'j}]_{n\times n}=\begin{bmatrix}- & \lambda_{1,2} & \cdots & \lambda_{1,n}\\ \lambda_{2,1} & - & \cdots & \lambda_{2,n}\\ \cdots & \cdots & \ddots & \cdots\\ \lambda_{n,1} & \lambda_{n,2} & \cdots & -\end{bmatrix}$, where $\lambda_{j'j}$ represents the number of times that job j appears immediately after job j'.

Step 3: Count the number of elements that are larger than zero in $[\phi_{i,j}]_{n\times n}$, and denote it as α.

Step 4: Count the number of elements that are larger than zero in $[\lambda_{jj'}]_{n\times n}$, and denote it as β.

Step 5. The diversity value of the population *div* is then computed as follows:

$$div=\left(\frac{\alpha-n}{n\times\min(n,x-1)}+\frac{\beta-(n-1)}{(n-1)\times\min(n-1,x-1)}\right)\Big/2,$$

where x is the population size. The above process is repeated until a termination condition is reached. pILS is outlined in Fig. 4.

3.3. Iterated Greedy methods: IGA and pIGA

IG was introduced by Ruiz and Stützle (2007) for solving the permutation flowshop with makespan criterion. IG starts from an initial solution generated by a heuristic and iterates over a main loop consisting of two phases: destruction and construction.

procedure The presented pILS algorithm
  Set the parameters x *and* γ
  Generate an initial population of size x using $LR(x)$
  $\pi^* \leftarrow$ *best solution in the population* % Best solution found so far
  repeat
    Select a solution π *from the population using the presented bi-selection scheme*
    $\pi' \leftarrow Perturbation(\pi)$ % Perform perturbation procedure
    $\pi'' \leftarrow iRZ(\pi')$ % Perform local search
    Accept π'' *according to the presented generational scheme*
      if $\sum C_j(\pi'') < \sum C_j(\pi^*)$ **then** % check if new best solution
        $\pi^* \leftarrow \pi''$
    endif
  until *termination criterion is met*
end

Fig. 4. Pseudo-code of the population extension of ILS or pILS.

During the destruction phase, some jobs are randomly removed from the current solution. Afterwards, the construction procedure applies a greedy constructive algorithm to reconstruct a complete solution by reinserting the previously removed jobs. Before continuing with the next iteration, an acceptance criterion decides whether the newly constructed solution replaces the incumbent solution. A local search is optionally applied to the initial solution and to the constructed solution. The procedure of the presented IG, referred to as IGA, is outlined in Fig. 5.

As can be seen, IGA can be considered as a variation of the basic ILS algorithm. The main difference is that ILS randomly perturbs a solution and in the IGA, this perturbation is carried out by a

procedure IGA
  $\pi_0 \leftarrow GenerateInitialSoulution$ % Generate an initial solution
  $\pi \leftarrow LocalSearch(\pi_0)$ % Local search
  repeat
    $\pi' \leftarrow Destruction\_Construction(\pi)$ % Destruction and construction
    $\pi'' \leftarrow LocalSearch(\pi')$ % Local search
    $\pi \leftarrow Acceptcrit erion(\pi'', \pi)$ % Decide if new solution replaces the incumbent
  until termination criterion is met
end

Fig. 5. Pseudo-code of the Iterated Greedy Algorithm (IGA) of Ruiz and Stützle (2007).

Procedure $Destruction\_Construction(\pi, d)$
  Set π_R empty, $\pi' \leftarrow \pi$ %Destruction
  for $i \leftarrow 1$ **to** d **do**
    $\pi' \leftarrow$ *remove a randomly selected job from* π'
    $\pi^R \leftarrow$ *include the removed job into* π^R
  endfor
  for $j \leftarrow 1$ **to** d **do** %Construction
    $\pi' \leftarrow$ *best permutation obtained after inserting job* π_j^R *in all possible positions of* π'
  endfor
end

Fig. 6. The destruction and construction procedure of Ruiz and Stützle (2007).

procedure the presented IGA
  Set the parameters d *and* λ
  $\pi_0 \leftarrow LR(n/m)$ % Generate an initial solution
  $\pi \leftarrow iRZ(\pi_0)$ % Local search
  $\pi^* \leftarrow \pi$ % Best solution found so far
  repeat
    $\pi' \leftarrow Destruction\_Construction(\pi)$ % Destruction and construction
    $\pi'' \leftarrow iRZ(\pi')$ % Local search
    if $\sum C_j(\pi'') < \sum C_j(\pi)$ **then** % Acceptance criterion
      $\pi \leftarrow \pi''$
      if $\sum C_j(\pi'') < \sum C_j(\pi^*)$ **then** % Check if new best solution
        $\pi^* \leftarrow \pi''$
      endif
    elseif $rand() \le \exp\{(\sum C_j(\pi) - \sum C_j(\pi''))/Temperature\}$ **then**
      $\pi \leftarrow \pi''$
    endif
  until *termination criterion is met*
end

Fig. 7. Pseudo-algorithm of the presented IGA.

destruction of the solution followed by a greedy reconstruction. If the greedy method is effective, IGA can outperform ILS, as shown in Ruiz and Stützle (2007) and others. The presented IGA adopts LR (n/m) to generate an initial solution, and employs iRZ as the local search procedure. Additionally, the same *Destruction_Construction* procedure as presented in Ruiz and Stützle (2007) is employed, where d jobs are randomly selected and removed and they are later inserted in all possible positions, one by one, in the construction procedure. The parameter d needs careful calibration. Finally, we employ the same *simulated annealing type* acceptance criterion as in the proposed ILS. The *Destruction_Construction* procedure is detailed in Fig. 6 (Ruiz and Stützle, 2007). The complete procedure of the presented IGA is described in Fig. 7.

Note that Pan et al. (2008) also proposed an IGA method for the PFSP and total flowtime minimization, denoted in this paper as IG_{RLS}. However, the authors employed a complex referenced local search method as well as some other added complexities. In comparison, our presented IGA is simpler and easier to code. As done with the pILS algorithm, the proposed IGA is extended in an identical way to form what we have denoted as pIGA. The same bi-tournament selection and generational scheme operators are employed.

4. Calibration of the proposed algorithms

ILS, pILS, IGA and pIGA have relatively few parameters, especially when compared to recently published metaheuristics. Still, these have to be properly calibrated. We employ a Design of Experiments (DOE, Montgomery, 2009) approach. DOE is an advanced statistical technique that helps in understanding the effect that some factors have over a given response variable. In our case, the factors are the parameters that need calibration and the response variable is the performance of the different algorithm configurations. Factors are tested at some levels or variants and therefore, some initial runs are required in order to pick a suitable set of levels to test. After a series of preliminary experiments, we consider the following levels for the parameters. For the ILS algorithm: perturbation length (γ) is tested at three levels: 2, 3 and 4; temperature factor (λ) is tested at four levels: 1.0, 2.0, 3.0 and 4.0. For the pILS algorithm, perturbation length (γ) is tested at three levels: 2, 3 and 4 and population size (x) is tested at four levels: 3, 5, 7 and 9. For IGA, destruction size (d) is tested at three levels: 6, 8 and 10 and the temperature factor (λ) is tested at four levels: 1.0, 2.0, 3.0 and 4.0. For pIGA, destruction size (d) is tested at three levels: 6, 8 and 10; population size (x) is tested at four levels: 3, 5, 7 and 9. We obtain a total of $3 \times 4 = 12$ different combinations, i.e., 12 different configurations for each of the proposed algorithms after combining all possible values of the tested factor levels. All the configurations of each algorithm are tested independently in a full factorial experimental design with a termination criterion set to a maximum elapsed CPU time $t = 10nm$ milliseconds. Note that this termination criterion increases with the size of the instance. This is needed in order to decouple the effect of the running time from the size of the instances, i.e., worse results could be wrongly attributed to the size of the instance instead of insufficient CPU time.

Each algorithm is tested with a set of 28 randomly generated instances. It is of paramount importance to separate the calibration benchmark from the final testing benchmark. Calibrating algorithms with the same benchmark results in over calibration and in too optimistic results, where those excellent results might not be transferrable to real instances or to other benchmarks. The number of jobs and machines for each calibration instance is randomly chosen from the following sets $n \in \{50, 75, 100, 125, 150, 175, 200\}$ and $m \in \{5, 10, 15, 20\}$. The processing times for each instance are obtained from discrete uniform distribution in the interval [1, 99]. For each instance, five independent replications are carried out in the experiments (i.e., each algorithm is run five times for each instance). Therefore, the total number of results is $12 \times 28 \times 5 = 1680$ for each one of the four presented algorithms. All the presented algorithms are coded in Visual C++ 6.0 and all the configurations are run on a cluster of 30 blade servers each one with two Intel XEON 5254 processors running at 2.5 GHz with 16 GB of RAM memory. There is no parallel computing. The 30 blade servers are just used in order to divide the workload and experimentations. As a response variable for the experiments, we calculate the relative percentage deviation (RPD) from a reference solution as follows:

$$RPD(c_i) = (c_i - c^*)/c^* \times 100 \tag{4}$$

where c_i is the total flowtime value generated in the ith replication by a given algorithm configuration, and c^* is the minimum total flowtime value found by any of the algorithm configurations. All the results are analyzed by means of a multi-factor Analysis of Variance (ANOVA) statistical technique where n and m are considered as non-controllable factors. This method has been used in Ballestín et al. (2007), Ruiz and Stützle (2007), Vallada and Ruiz (2010), and many others. ANOVA is a very powerful statistical approach that allows setting the different parameters at statistically significant values among the tested ones. ANOVA is a parametric test and it is needed to check its three main hypotheses, i.e., normality, homogeneity of variance (or homoscedasticity) and independence of the residuals. Given the large number of treatments and replicates, the residual analysis showed no indication of severe violation of any of the hypotheses.

Due to reasons of space, we briefly comment the results of the ANOVA analysis and calibration. For the ILS algorithm, the perturbation length (γ) results in statistically significant differences in the response variable at a 95% confidence level, whereas the temperature factor (λ) does not yield significant differences (this is consistent with the results of Stützle, 2006 and Ruiz and Stützle, 2007). This suggests that the ILS algorithm is robust with respect to the temperature factor, at least with the tested values ($\lambda \in \{1, 2, 3, 4\}$). For the pILS algorithm, both factors (γ and x) are statistically significant. For IGA, the destruction size (d) is significant while the temperature factor (λ) is not (again, this is consistent with the calibrations given in Ruiz and Stützle, 2007). For pIGA, population size (x) results in significant differences, while the destruction size (d) does not. After the calibration experiments, we set the parameters as follows. For the ILS algorithm, $\gamma = 2$ and $\lambda = 4.0$. For the pILS algorithm, $\gamma = 2$ and $x = 3$. IGA: $d = 8$ and $\lambda = 2.0$ and pIGA, $d = 8$ and $x = 3$. All experimental results, tables and plots are available upon request from the authors.

5. Computational and statistical evaluation

We now compare the four proposed methods against the best algorithms from the literature. For the evaluation we employ the well known benchmark of Taillard (1993). This test bed has been used in Liu and Reeves (2001), Tasgetiren et al. (2007), Tseng and Lin (2009), and almost in every PFSP paper. There are a total of 120 instances where $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. These instances are divided into 12 subsets, resulting from the combinations of values for n and m. There are 10 replicates in each subset. Not all combinations are present and the sets available are 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20 and 500×20. To maintain the orthogonality in the experiment, we generate the three missing additional subsets of instances: 200×5, 500×5 and 500×10. These are extracted from instances 200×10 and 500×20. We take the processing times of the first five machines of instances

in subset 200×10 to create instances of the subset 200×5, and extract the processing times of the first five or 10 machines of instances in subset 500×20 to generate the instances of the subsets 500×5 or 500×10, respectively. In total, we use 150 instances for each algorithm.

We re-implemented 12 powerful metaheuristics presented in recent years, and compare them with the algorithms of this paper. The algorithms implemented are: DDE_{RLS} and IG_{RLS} of Pan et al. (2008), HGA_{T1} of Tseng and Lin (2009), HGA_{T2} of Tseng and Lin (2010), HGA_Z of Zhang et al. (2009), ILS_D of Dong et al. (2009), EDA_J and VNS_J of Jarboui et al. (2009), AGA of Xu et al. (2011), DABC and hDDE of Tasgetiren et al. (2011) and SLS of Dubois-Lacoste et al. (2011). Dubois-Lacoste et al. (2011), presented an Iterated Greedy Algorithm for the bi-objective flowshop. However, and although not tested in the original paper, a simpler method is proposed for the total flowtime flowshop. Therefore, we also test it in this paper.

All algorithms have been coded in Visual C++ 6.0. We strictly follow all original explanations and details given in the original papers in order to closely reproduce published results. All methods are run on a cluster of 30 blade severs each one with two Intel XEON 5254 quad core processors running at 2.5 GHz with 16 GB of RAM memory. The experiments are carried out in virtualized Windows XP machines, each one with one virtualized processor and 2 GB of RAM memory. To make a fair comparison, all the algorithms adopt the same maximum elapsed CPU time limit of $t = \rho mn$ milliseconds as a termination criterion, where ρ has been tested at three values: 30, 60 and 90. The choice of this stopping criterion is motivated by the fact that all algorithms are coded in the same programming language, share most library functions and data structures, and are executed on the same computer environment. Then we can safely say that all algorithms have the same CPU power and time available and that results are fully comparable. This termination criterion has been increasingly used in the recent literature on scheduling Ruiz et al. (2006), Ruiz and Stützle (2007, 2008), Vallada and Ruiz (2010), Ribas et al. (2011) and several others. Additionally, with the three termination criteria, we can test how the different algorithms perform with different CPU times. $\rho = 30$ turns into three seconds for the smallest instances of 20×5 whereas $\rho = 90$ translates into 900 s for the largest instances of 500×20. Therefore, in the tests we run all methods from small to large CPU times. For each of the 150 instances, five independent runs are carried out for each algorithm. We calculate the average relative percentage deviation from the best known solution for each instance. The computed results, averaged across the five replications for each instance and grouped for each subset, are reported in Tables 1–3.

From the computational results all of the presented algorithms yield solutions that are most of the time better than those of the other methods. From Table 1 where $\rho = 30$, we can see that the largest overall average RPD (AVRPD) value generated by the

Table 1
Computational results of the algorithms ($\rho = 30$). Best and worst values in bold and italics, respectively.

Instances	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}	AGA	hDDE	DABC	SLS	IGA	pIGA	ILS	pILS
20×5	0.01	0.01	1.32	*2.92*	0.01	0.15	0.01	0.02	0.01	0.01	**0.00**	**0.00**	**0.00**	0.01	**0.00**	0.01
20×10	**0.00**	**0.00**	1.73	*2.90*	**0.00**	0.24	0.02	0.04	0.03	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.01
20×20	**0.00**	**0.00**	1.06	*2.15*	**0.00**	0.13	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50×5	0.35	0.35	3.06	*3.08*	0.90	1.42	**0.29**	0.92	0.85	0.40	0.71	0.73	0.34	0.38	0.38	0.42
50×10	0.45	0.48	4.19	*4.38*	0.93	2.02	0.58	0.97	1.33	0.68	0.82	0.40	0.40	**0.37**	0.42	0.39
50×20	0.52	0.52	4.11	*4.53*	0.77	1.93	0.62	1.00	1.34	0.73	0.81	**0.34**	0.50	0.46	0.47	0.44
100×5	0.35	0.33	*6.65*	3.01	0.62	2.13	0.56	3.69	0.68	0.44	0.93	0.74	**0.24**	0.26	0.31	0.33
100×10	0.50	0.52	*7.82*	4.54	0.98	2.77	1.49	5.01	1.66	1.04	1.69	0.52	0.46	**0.38**	0.50	0.42
100×20	0.69	0.67	*7.02*	4.72	1.02	2.98	2.05	4.88	2.16	1.10	1.68	**0.49**	0.60	0.56	0.52	0.59
200×5	0.28	0.31	*12.18*	4.01	0.24	2.33	0.48	8.74	0.40	0.48	0.67	0.64	**0.11**	0.12	0.16	0.22
200×10	0.46	0.45	*12.06*	5.79	0.46	3.01	1.29	9.56	1.09	1.12	1.24	0.53	**0.27**	0.37	**0.27**	0.50
200×20	0.67	0.66	*11.41*	6.35	0.67	3.22	2.21	8.44	2.24	1.62	1.89	0.43	**0.37**	0.63	0.40	0.53
500×5	0.24	0.26	*15.96*	7.77	**0.09**	3.32	0.16	13.74	0.16	0.34	0.32	0.47	0.13	**0.09**	0.13	0.14
500×10	0.55	0.56	*14.15*	8.49	0.17	3.92	0.37	12.17	0.31	0.73	0.68	0.51	**0.07**	0.16	0.08	0.24
500×20	0.75	0.69	*13.14*	8.48	0.48	3.84	0.93	10.12	0.77	0.94	0.99	0.37	**0.14**	0.42	0.16	0.45
Average	0.39	0.39	*7.72*	4.88	0.49	2.23	0.74	5.29	0.87	0.64	0.83	0.41	**0.24**	0.28	0.25	0.31

Table 2
Computational results of the algorithms ($\rho = 60$). Best and worst values in bold and italics, respectively.

Instances	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}	AGA	hDDE	DABC	SLS	IGA	pIGA	ILS	pILS
20×5	0.01	0.01	1.23	*2.92*	0.01	0.12	0.01	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	0.01
20×10	**0.00**	**0.00**	1.59	*2.90*	**0.00**	0.21	0.02	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
20×20	**0.00**	**0.00**	1.03	*2.15*	**0.00**	0.10	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50×5	0.35	0.34	2.67	*3.14*	0.85	1.19	**0.18**	0.78	0.61	0.39	0.58	0.72	0.35	0.36	0.36	0.39
50×10	0.44	0.47	3.90	*4.47*	0.90	1.85	0.38	0.80	1.06	0.61	0.66	**0.36**	0.43	0.41	0.42	0.39
50×20	0.48	0.51	3.93	*4.57*	0.67	1.76	0.40	0.72	1.05	0.63	0.64	**0.24**	0.47	0.41	0.45	0.42
100×5	0.33	0.29	*5.02*	2.70	0.68	2.01	0.42	2.25	0.63	0.41	0.90	0.80	**0.25**	0.26	0.33	0.33
100×10	0.48	0.49	*6.23*	4.20	0.94	2.75	1.14	2.78	1.46	0.88	1.57	0.57	0.41	**0.37**	0.50	0.40
100×20	0.65	0.71	*5.93*	4.43	1.00	3.02	1.58	2.60	1.86	0.99	1.52	**0.49**	0.62	0.51	0.51	0.53
200×5	0.24	0.26	*10.87*	3.19	0.32	2.35	0.44	6.69	0.43	0.38	0.45	0.69	**0.15**	**0.15**	0.19	0.25
200×10	0.42	0.38	*10.97*	5.02	0.51	2.97	1.19	8.10	1.10	1.02	1.22	0.51	**0.23**	0.35	0.27	0.45
200×20	0.59	0.57	*10.42*	5.57	0.63	3.26	2.02	7.51	2.15	1.52	1.94	0.45	0.37	0.50	**0.34**	0.47
500×5	0.22	0.22	*15.16*	5.95	0.11	3.10	0.19	13.39	0.16	0.36	0.32	0.49	**0.07**	0.09	0.08	0.14
500×10	0.51	0.49	*13.67*	7.16	0.20	3.64	0.41	11.86	0.34	0.74	0.65	0.48	**0.08**	0.16	0.10	0.23
500×20	0.70	0.64	*12.71*	7.41	0.49	3.57	1.05	9.96	0.87	1.02	1.00	0.29	**0.19**	0.47	0.22	0.48
Average	0.36	0.36	*7.02*	4.39	0.49	2.13	0.63	4.50	0.78	0.60	0.76	0.41	**0.24**	0.27	0.25	0.30

Table 3
Computational results of the algorithms ($\rho = 90$). Best and worst values in bold and italics, respectively.

Instances	IG_{RLS}	DDE_{RLS}	EDA_J	VNS_J	ILS_D	HGA_{T1}	HGA_Z	HGA_{T2}	AGA	hDDE	DABC	SLS	IGA	pIGA	ILS	pILS
20×5	0.01	0.01	1.20	*2.92*	0.01	0.09	0.01	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	0.01
20×10	**0.00**	**0.00**	1.56	*2.90*	**0.00**	0.15	0.02	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
20×20	**0.00**	**0.00**	0.97	*2.15*	**0.00**	0.06	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50×5	0.34	0.32	2.58	*3.16*	0.85	1.07	**0.17**	0.68	0.49	0.37	0.49	0.70	0.31	0.33	0.31	0.34
50×10	0.47	0.47	3.87	*4.53*	0.86	1.76	0.39	0.73	0.90	0.60	0.62	**0.36**	0.44	0.41	0.42	0.40
50×20	0.51	0.53	3.91	*4.64*	0.69	1.68	0.42	0.69	0.92	0.63	0.64	**0.24**	0.50	0.43	0.46	0.44
100×5	0.29	0.26	*4.18*	2.63	0.71	1.93	0.31	1.66	0.52	0.37	0.83	0.80	**0.23**	0.24	0.33	0.31
100×10	0.54	0.52	*5.57*	4.20	1.01	2.79	0.96	2.23	1.34	0.87	1.50	0.63	0.45	**0.41**	0.53	0.44
100×20	0.63	0.68	*5.46*	4.42	1.01	2.98	1.35	1.99	1.70	0.94	1.37	**0.47**	0.59	0.52	0.48	0.48
200×5	0.20	0.23	*10.12*	2.75	0.36	2.31	0.40	5.72	0.41	0.33	0.44	0.72	**0.14**	0.15	0.20	0.26
200×10	0.39	0.37	*10.16*	4.63	0.57	2.99	1.13	7.24	1.10	0.98	1.22	0.51	**0.24**	0.31	0.29	0.44
200×20	0.53	0.53	*9.63*	5.13	0.63	3.30	1.91	6.52	1.97	1.46	1.94	0.41	0.35	0.45	**0.33**	0.40
500×5	0.21	0.21	*14.69*	5.17	0.12	3.08	0.20	12.71	0.17	0.36	0.31	0.50	**0.08**	0.10	0.09	0.14
500×10	0.43	0.43	*13.26*	6.47	0.21	3.59	0.42	11.44	0.34	0.72	0.70	0.45	**0.08**	0.16	0.11	0.23
500×20	0.65	0.61	*12.45*	6.83	0.49	3.58	1.12	9.77	0.93	1.05	1.08	0.27	**0.22**	0.48	0.25	0.47
Average	0.35	0.34	*6.64*	4.17	0.50	2.09	0.59	4.09	0.72	0.58	0.74	0.40	**0.24**	0.27	0.25	0.29

presented algorithms is 0.31%, which is much smaller than those of the competing methods, being IG_{RLS} and DDE_{RLS} the closest competitors with average deviations of 0.39%. IGA is the best performer with an AVRPD value equal to 0.24%, followed by ILS (0.25%), pIGA (0.28%) and pILS (0.31%). Of special interest is comparing ILS_D with the proposed ILS, as both methods are based on the ILS framework. We can see that our proposed ILS gives results that are lower on average than ILS_D. Also, when comparing IG_{RLS} with our proposed IGA we see that the results of IGA are, on average, lower than those of IG_{RLS}. Other methods, such as SLS, provide the best solutions for some specific instance sizes (50×20 and 100×20). As we can see, pIGA does not manage to outperform the simpler IGA. The same can be said about pILS when compared to ILS. However, for some specific instance groups (100×10), it seems that the population methods achieve a slightly better performance. This better performance is later shown to be statistically significant. In any case, the added complexity of pILS and pIGA does not seem worth given the marginally worse results. It has to be noted that earlier versions of pILS and pIGA without the bi-selection scheme and diversity control mechanism were clearly worse than ILS and IGA. Therefore, it seems clear that simple methods like ILS and IGA that iterate over a single solution, work best.

For $\rho = 60$ and $\rho = 90$, we again find from Tables 2 and 3 that the results of the proposed algorithms are much lower than those of the others and IGA is again the best performing method in terms of AVRPD. We also see how the four proposed methods barely improve from one table to the other, meaning that all four converge rapidly and additional CPU time does not translate into much better solutions. The same can be said about the best performing competing methods such as IG_{RLS}, DDE_{RLS}, SLS and ILS_D, i.e., results improve only slightly with double and triple allowed CPU time. However, for the other methods, larger improvements are seen with additional CPU time but these are not enough to compete with the best methods. Hence, it can be concluded that the proposed local search based algorithms perform better than the 12 compared competing methods for the problem considered and under our experimental settings.

It is worth insisting that there are many similarities in the eight best performing algorithms with AVRPD values of less than 0.5%. Namely, the LR heuristic is used most of the times to generate initial solutions, and the RZ improvement procedure is used as a local search phase also in most of these high performing algorithms. This evidences the effectiveness of taking advantage of the LR heuristic and the RZ-improvement procedure for solving the PFSP with total flowtime criterion. In other words, the relatively worse performance of EDA_J, HGA_{T1}, HGA_{T2} and others might be mainly due to their relatively worse initialization and due to the less effective local search methods. In any case, our proposed methods are arguably simpler than most others and still attain the best performance. The superiority of the presented algorithms and the ILS_D of Dong et al. (2009) demonstrates the effectiveness of simple local search frameworks. Together with the fact that local search also plays a significant role in most high performing methods, we conclude that a well designed local search based algorithm is all that is needed in order to obtain state-of-the-art results for the problem considered without turning into more complex methods such as genetic or estimation of distribution algorithms.

To check whether the observed differences from the above Tables 1–3 are indeed statistically significant, we carry out a multifactor statistical ANOVA test where n, m, replication, CPU time parameter ρ and the type of algorithm are considered as factors. We compare the twelve best performing algorithms only: IGA, pIGA, ILS, pILS, DDE_{RLS}, IG_{RLS}, SLS, ILS_D, hDDE, HGA_Z, DABC and AGA. The remaining four algorithms (HGA_{T1}, HGA_{T2}, EDA_J and VNS_J) were ruled out since it was not needed to test for significance, since their results were clearly worse than the rest. The ANOVA results (not shown in detail due to reasons of space) indicate that n, m, ρ and the type of algorithm result in statistically significant differences in the response variable RPD at a 95% confidence level, whereas the replication does not show significant differences (replicate is not expected to be significant, so this outcome validates the statistical test. These factors are often referred to as *witness factors*). Fig. 8 reports the means plot together with 95% Tukey honest significant differences (HSD) confidence intervals of the interaction between the type of algorithms and CPU time parameter ρ. Note that overlapping intervals denote statistically insignificant differences between the plotted overlapped means. Each plotted average corresponds to the average of 150 instances run five times (750 results). HSD confidence intervals are conservative and counter the bias in the type I statistical error of multiple pairwise comparisons. The figure depicts the overall mean, without separating each instance size, "zoomed-in" figures for each instance size show slightly different results and in some occasions, not so wide intervals. From the figure it is clear that the results of IGA and ILS are statistically better than those of competing methods. pIGA and pILS are, on average, statistically equivalent to DDE_{RLS} and IG_{RLS}. They are, however, statistically better than the next best competing method (SLS). In the figure we can see that the better the method, the smaller the difference in the means as CPU time increases. For $\rho = 60$ and $\rho = 90$, i.e., double or triple the CPU time, IGA shows a complete overlap of the three intervals. However, pILS shows a slight improvement as CPU time increases

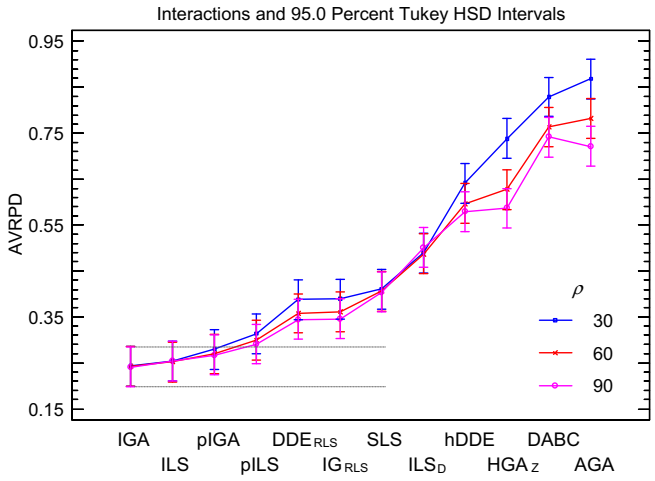

Fig. 8. Means and confidence intervals of the interaction between the best tested algorithms and the allowed CPU time in the ANOVA experiment.

but this improvement is far from being statistically significant. Only the last three methods, HGA_Z, DABC and AGA, show statistically better results as ρ increases. This means that the algorithms have not converged and that they might need a substantial additional CPU time to reach the results of the other methods. As a final conclusion, we can safely state that the proposed algorithms are the best performers for the permutation flowshop scheduling problem with the objective of minimizing total flowtime. Arguably, we can also state that IGA and ILS are markedly simple and easier to implement than some of the other methods.

In order to facilitate follow up research, we report the best known solutions found so far in Table 4. This was already done in Jarboui et al. (2009) and Xu et al. (2011), among others. We run our algorithms for a maximum elapsed CPU time $t = 400mn$ milliseconds. We compare the best solution found by our algorithms with the solutions reported by Pan et al. (2008), Tseng and Lin (2009), Zhang et al. (2009), Dong et al. (2009), Jarboui et al. (2009) and Tseng and Lin (2010), Czapiński (2010), Zheng and Yamashiro (2010), Zhang and Li (2011), Tasgetiren et al. (2011), Xu et al. (2011). The best solution for each of the 120 Taillard (1993) instances is calculated by closely examining all existing results. For the new 30 instances generated from Taillard's instances, we show the best solution found by all the compared algorithms in this paper. We use $TA91_5$ to represent the instance obtained from TA91 by considering the processing times from the first five machines. It is interesting to see from Table 4 that the proposed algorithms in this paper have further improved 12 out of 120 instances. Note that these new 12 best solutions have been obtained for the largest and therefore presumably hardest instances of Taillard.

6. Conclusions

The permutation flowshop scheduling problem with total flowtime minimization has been subject of intense research in the last years. Complex and high performing metaheuristic algorithms have been introduced. In this paper, we have proposed four simple methods, including an iterated greedy algorithm (IGA), an Iterated Local Search (ILS), a population-based IGA (pIGA), and a population-based ILS (pILS). These algorithms perform an extensive search in the space of local optima. They are very simple, easy to implement and to replicate but at the same time they provide state-of-the-art results.

The best combination of parameters for each algorithm was obtained by means of a Design of Experiments approach that involves the evaluation of different alternatives. The evaluation of the proposed methods was carried out against the 12 best performing

Table 4
Best known total flowtime values for Taillard benchmark instances.

Instance No.	Best solution	Instance No.	Best solution	Instance No.	Best solution	Instance No.	Best solution	Instance No.	Best solution
TA01	14033	TA11	20911	TA21	33623	TA31	64802	TA41	87114
TA02	15151	TA12	22440	TA22	31587	TA32	68051	TA42	82820
TA03	13301	TA13	19833	TA23	33920	TA33	63162	TA43	79931
TA04	15447	TA14	18710	TA24	31661	TA34	68226	TA44	86446
TA05	13529	TA15	18641	TA25	34557	TA35	69351	TA45	86377
TA06	13123	TA16	19245	TA26	32564	TA36	66841	TA46	86587
TA07	13548	TA17	18363	TA27	32922	TA37	66253	TA47	88750
TA08	13948	TA18	20241	TA28	32412	TA38	64332	TA48	86727
TA09	14295	TA19	20330	TA29	33600	TA39	62981	TA49	85441
TA10	12943	TA20	21320	TA30	32262	TA40	68770	TA50	87998
TA51	125831	TA61	253266	TA71	298385	TA81	365463	TA91	1046314
TA52	119247	TA62	242281	TA72	274384	TA82	372449	TA92	1034195
TA53	116459	TA63	237832	TA73	288114	TA83	370027	TA93	1046902
TA54	120261	TA64	227738	TA74	301044	TA84	372393	TA94	1030481
TA55	118184	TA65	240301	TA75	284681	TA85	368915	TA95	1034027
TA56	120586	TA66	232342	TA76	269686	TA86	370908	TA96	1006195
TA57	122880	TA67	240366	TA77	279463	TA87	373408	TA97	1053051
TA58	122489	TA68	230945	TA78	290908	TA88	384525	TA98	1044875
TA59	121872	TA69	247921	TA79	301970	TA89	374423	TA99	1026137
TA60	123954	TA70	242933	TA80	291283	TA90	379296	TA100	1030299
TA101	**1227733**	TA111	**6698656**	$TA91_5$	937273	$TA111_5$	5539387	$TA111_{10}$	5997531
TA102	1245271	TA112	6770735	$TA92_5$	896936	$TA112_5$	5608131	$TA112_{10}$	6106675
TA103	**1269673**	TA113	6739645	$TA93_5$	936905	$TA113_5$	5605732	$TA113_{10}$	6073492
TA104	**1238349**	TA114	**6785991**	$TA94_5$	902818	$TA114_5$	5526960	$TA114_{10}$	6062847
TA105	**1227214**	TA115	**6729468**	$TA95_5$	920723	$TA115_5$	5588103	$TA115_{10}$	5986526
TA106	1227604	TA116	**6724085**	$TA96_5$	890028	$TA116_5$	5497811	$TA116_{10}$	6006542
TA107	1243707	TA117	6691468	$TA97_5$	930040	$TA117_5$	5483350	$TA117_{10}$	5966581
TA108	**1246123**	TA118	**6783916**	$TA98_5$	914638	$TA118_5$	5572833	$TA118_{10}$	6080320
TA109	**1234936**	TA119	6711305	$TA99_5$	910726	$TA119_5$	5554145	$TA119_{10}$	5994142
TA110	1250596	TA120	**6755722**	$TA100_5$	903188	$TA120_5$	5509152	$TA120_{10}$	6013461

Bold values represent the new best known solutions found by the proposed algorithms in this paper.

methods from the literature. According to the extensive experimental and statistical analyses, the proposed IGA and ILS methods performed better than pIGA and pILS, and they outperform the existing methods for the problem considered. The fact that simple methods perform better than complex existing approaches, and also that for our proposed heuristics using populations did not improve results, reinforces the idea that simple local search based methods are enough to solve the PFSP with total flowtime criterion.

After comparing the best solutions produced by the presented algorithms and those reported in the literature, we found that 12 out of 120 best known solutions for Taillard's benchmark suite were further improved by the presented algorithms.

Future research directions involve the consideration of more complex scheduling problems and objectives. It seems worthwhile to apply the presented algorithms to more realistic scheduling problems like those with setup times, parallel machines, buffer size constraints, no-idle and no-wait considerations. There have been already many studies in this regard and it is possible that simple methods also perform equally well in those settings.

Acknowledgements

This research is partially supported by National Science Foundation of China under Grants 61174187, 60874075, and Basic scientific research foundation of Northeast University under Grant N110208001, and Science Foundation of Shandong Province, China (BS2010DX005). Rubén Ruiz is partially funded by the Spanish Ministry of Science and Innovation, under the project "SMPA – Advanced Parallel Multiobjective Sequencing: Practical and Theoretical Advances" with reference DPI2008-03511/DPI, by the Small and Medium Industry of the Generalitat Valenciana (IMPIVA) and by the European Union through the European Regional Development Fund (FEDER) inside the R + D program "*Programa de I + D para Institutos Tecnológicos de la Red IMPIVA*" during the year 2010, with Project Number IMDEEA/2011/142.

References

Allahverdi, A., Aldowaisan, T., 2002. New heuristics to minimize total completion time in *m*-machine flowshops. International Journal of Production Economics 77 (1), 71–83.

Baker, K.R., 1974. Introduction to Sequencing and Scheduling. John Wiley & Sons, New York.

Ballestín, F., Schwindt, C., Zimmermann, J., 2007. Resource leveling in make-to-order production: modeling and heuristic solution method. International Journal of Operations Research 4 (1), 50–62.

Bansal, S.P., 1977. Minimizing the sum of completion times of *n* jobs over *m* machines in a flowshop: a branch and bound approach. IIE Transactions 9 (3), 306–311.

Czapiński, M., 2010. Parallel simulated annealing with genetic enhancement for flowshop problem with C_{sum}. Computers & Industrial Engineering 59 (4), 778–785.

Dong, X.Y., Huang, H.K., Chen, P., 2009. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. Computers & Operations Research 36 (5), 1664–1669.

Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T., 2011. A hybrid TP + PLS algorithm for bi-objective flow-shop scheduling problems. Computers & Operations Research 38 (8), 1219–1236.

Fanjul-Peyro, L., Ruiz, R., 2010. Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 207 (1), 55–69.

Framinan, J.M., Leisten, R., Ruiz-Usano, R., 2005. Comparison of heuristics for flowtime minimisation in permutation flowshops. Computers & Operations Research 32 (5), 1237–1254.

Gonzalez, T., Sahni, S., 1978. Flowshop and jobshop schedules: complexity and approximation. Operations Research 26 (1), 36–52.

Gupta, J.N.D., 1972. Heuristic algorithms for multistage flowshop scheduling problem. AIIE Transactions 4 (1), 11–18.

Gupta, J.N.D., Chen, C.L., Yap, L.Y., Deshmukh, H., 2000. Designing a tabu search algorithm to minimize total flow time in a flow shop. Arabian Journal for Science and Engineering 25 (1C), 79–94.

Ignall, E., Schrage, L.E., 1965. Application of the branch and bound technique to some flow shop scheduling problems. Operations Research 13 (3), 400–412.

Jarboui, B., Ibrahim, S., Siarry, P., Rebai, A., 2008. A combinatorial particle swarm optimization for solving permutation flowshop problems. Computers & Industrial Engineering 54 (3), 526–538.

Jarboui, B., Eddaly, M., Siarry, P., 2009. An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. Computers & Operations Research 36 (9), 2638–2646.

Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1 (1), 61–68.

Laha, D., Sarin, S.C., 2009. A heuristic to minimize total flow time in permutation flow shop. OMEGA, The International Journal of Management Science 37 (3), 734–739.

Laurent, B., Hao, J.-K., 2009. Iterated local search for the multiple depot vehicle scheduling problem. Computers & Industrial Engineering 57 (1), 277–286.

Li, X., Wang, Q., Wu, C., 2009. Efficient composite heuristics for total flowtime minimization in permutation flow shops. OMEGA, The International Journal of Management Science 37 (1), 155–164.

Li, X., Wu, C., 2005. An efficient constructive heuristic for permutation flow shops to minimize total flowtime. Chinese Journal of Electronics 14 (2), 203–208.

Liu, J., Reeves, C.R., 2001. Constructive and composite heuristic solutions to the $P//\sum C_j$ scheduling problem. European Journal of Operational Research 132 (2), 439–452.

Lourenço, H.R., Martin, O.C., Stützle, T., 2010. Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics, . second ed., International Series in Operations Research & Management Science second ed., vol. 14 Kluwer Academic Publishers, Norwell, MA, pp. 363–397 (Chapter 12).

Minella, G., Ruiz, R., Ciavotta, M., 2011. Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems. Computers & Operations Research 38 (11), 1521–1533.

Montgomery, D., 2009. Design and Analysis of Experiments, seventh ed.. John Wiley & Sons, New York.

Osman, I.H., Potts, C.N., 1989. Simulated annealing for permutation flow-shop scheduling. OMEGA, The International Journal of Management Science 17 (6), 551–557.

Pan, Q.-K., Tasgetiren, M.F., Liang, Y.-C., 2008. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers & Industrial Engineering 55 (4), 795–816.

Pan, Q.-K., Ruiz, R., 2012. An estimation of distribution algorithm for lot-streaming flow shop problems with setup times OMEGA. The International Journal of Management Science 40 (2), 166–180.

Pinedo, M., 2009. Scheduling: Theory,, .. Algorithms and Systems, third ed. Springer, New York.

Rajendran, C., 1993. Heuristic algorithm for scheduling in flowshop to minimize total flowtime. International Journal of Production Economics 29 (1), 65–73.

Rajendran, C., Ziegler, H., 1997. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Journal of Operational Research 103 (1), 129–138.

Rajendran, C., Ziegler, H., 2004. Ant-colony algorithms for permutation flowhsop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 155 (2), 426–438.

Rajendran, C., Ziegler, H., 2005. Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. Computers & Industrial Engineering 48 (4), 789–797.

Ribas, I., Companys, R., Tort-Martorell, X., 2011. An iterated greedy algorithm for the flowshop scheduling problem with blocking. OMEGA, The International Journal of Management Science 39 (3), 293–301.

Ruiz, R., Maroto, C., Alcaraz, J., 2006. Two new robutst genetic algorithms for the flowshop scheduling problem. OMEGA, The International Journal of Management Science 34 (5), 461–476.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177 (3), 2033–2049.

Ruiz, R., Stützle, T., 2008. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research 187 (3), 1143–1159.

Stafford, Jr. E.F., 1988. On the development of a mixed integer linear programming model for the flowshop sequencing problem. Journal of the Operational Research Society 39 (12), 1163–1174.

Stützle, T., 1998. Local Search Algorithms for Combinatorial Problems – Analysis, Algorithms, and New Applications. PhD thesis. TU Darmstadt, Computer Science Department. Darmstadt, Germany.

Stützle, T., 1998b. Applying Iterated Local Search to the Permutation Flowshop Problem. Technical Report AIDA-98-04. FG Intellektik, TU Darmstadt. Darmstadt, Germany.

Stützle, T., 2006. Iterated local search for the quadratic assignment problem. European Journal of Operational Research 174 (3), 1519–1539.

Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research 47 (1), 65–74.

Taillard, E., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64 (2), 278–285.

Tang, L., Liu, J., 2002. A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time. Journal of Intelligent Manufacturing 13 (1), 61–67.

Tasgetiren, M.F., Liang, Y.-C., Sevkli, M., Gencyilmaz, G., 2007. A particle swarm optimization algorithm for makespan and total flowtime minimization in the

permutation flowshop sequencing problem. European Journal of Operational Research 177 (3), 1930–1947.
Tasgetiren, M.F., Pan, Q.-K., Suganthan, P.N., Chen, A.H.-L., 2011. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow Shops. Information Sciences 181 (16), 3459–3475.
Tseng, L.-Y., Lin, Y.-T., 2009. A hybrid genetic local search for the permutation flowshop scheduling problem. European Journal of Operational Research 198 (1), 84–92.
Tseng, L.-Y., Lin, Y.-T., 2010. A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. International Journal of Production Economics 127 (1), 121–128.
Urlings, T., Ruiz, R., Stützle, T., 2010. Shifting representation search for hybrid flexible flowline problems. European Journal of Operational Research 207 (2), 1086–1095.
Vallada, E., Ruiz, R., 2010. Genetic algorithm with path relinking for the minimum tardiness permutation flowshop problem. OMEGA, The International Journal of Management Science 38 (1-2), 556–575.
Vempati, V.S., Chen, C.-L., Bullington, S.F., 1993. An effective heuristic for flow shop problems with total flow time as criterion. Computers & Industrial Engineering 25 (1-4), 219–222.
Xu, X., Xu, Z., Gu, X., 2011. An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. Expert systems with Applications 38 (7), 7970–7979.
Yamada, T., Reeves, C.R., 1998. Solving the C_{sum} permutation flowshop scheduling problem by genetic local search. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, pp. 230–234.
Yao, J., Kharma, N., Grogono, P., 2010. Bi-objective Multipopulation genetic algorithm for multimodal function optimization. IEEE Transactions on Evolutionary Computation 14 (1), 80–102.
Zhang, Y., Li, X., Wang, Q., 2009. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. European Journal of Operational Research 196 (3), 869–876.
Zhang, Y., Li, X., 2011. Estimation of distribution algorithm for permutation flow shops with total flowtime minimization. Computers & Industrial Engineering 60 (4), 706–718.
Zheng, T., Yamashiro, M., 2010. Solving flow shop scheduling problems by quantum differential evolutionary algorithm. International Journal of Advanced Manufacturing Technology 49 (5-8), 643–662.